Aalto University
School of Science
Degree Programme of Computer Science and Engineering

Naufal Khalid

Highly Efficient Search In Linguistic Data

Master's Thesis
Espoo, April 10, 2018

Supervisors: Gionis, Aristides
Instructor: Gionis, Aristides;
Stén, Liisa

Aalto University
School of Science
Degree Programme of Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Naufal Khalid

Title:
Highly Efficient Search In Linguistic Data

Date:	April 10, 2018	**Pages:**	vi + 64
Professorship:	Computer Science	**Code:**	CS-E4420
Supervisors:	Gionis, Aristides		
Instructor:	Gionis, Aristides; Stén, Liisa		

Electronic dictionaries and online learning services have become a common tool for translators, linguistics and people trying to learn a new language. This master's thesis work has been carried out in the company Kielikone. The company owns an online learning service called Sanakirja.fi, which helps in studying and teaching foreign languages with the help of a dictionary search service, where people can make bidirectional searches for headwords in multiple languages (over 40). The goal of this thesis is to analyse and develop a unified interface to search entries from different dictionary resources.

In this master's thesis work we try to develop a 'standard' information retrieval system on linguistic data. Although the system uses many different methods existing in Information Retrieval Systems, it also contains facilities, optimized for linguistic data. We have tried to compare and use different techniques of information retrieval and learning to rank to store, retrieve and rank information from a very large data source, containing around 80 million records. The system also supports approximate search which has been implemented using wild card searching algorithm and ranks results according to the degree of closeness to the query.

In the end we implement a learning to rank system which learns from user feedback and the results are passed to a logistic regression classifier. The classifier predicts the feature coefficients and those, combined with the real weight features are used to improve the ranking of the system.

Keywords: information retrieval, electronic dictionaries, learning to rank, indexing, searching, ranking

Language: English

Acknowledgements

Firstly, I would like to offer my gratitude to Almighty ALLAH, the most Merciful, the most Gracious, the most Powerful for giving me the strength and motivation to complete this thesis.

I would like to thank my thesis supervisor Prof. Aristides Gionis for providing me guidance, assistance and timely feedback for this thesis.

I would also like to thank my company Kielikone Oy for giving me this opportunity to work on this thesis. I would also acknowledge that they allowed me to work part time so I can finish my thesis on time. I would also thank all my employees at Kielikone Oy for providing me the motivation and support throughout.

A special thanks to all my friends for they have always stood alongside me whenever I needed them. It was because of their continuous taunts, I had the motivation to complete my thesis on time

And Lastly, I would like to thank my parents and family for their endless support and prayers for me. It would never have been possible without your love, prayers and support.

Espoo, April 10, 2018

Naufal Khalid

Abbreviations and Acronyms

IR	Information Retrieval
IRS	Information Retrieval Systems
VSM	Vector Space Model
BRM	Boolean Retrieval Model
PRP.	Probability Ranking Principle
BIM	Binary Independence Model
IDF	Inverse Document Frequency
LTR	Learning to rank
TFIDF	Term Frequency Inverse Document Frequency

Contents

Chapter 1

Introduction

We are far away from those days where people used to search for word translations, using printed form of dictionaries. Electronic dictionaries and online learning services have become a common tool for translators, linguistics and people trying to learn a new language. They provide a single, efficient interface, to look up for entries, combining information from multiple sources and dictionaries. Considering the computing power of modern computers and highly efficient search algorithms, information can be retrieved with ease and efficiency.

1.1 Sanakirja.fi and Search Service

Sanakirja.fi is an online learning service which helps its users to learn and teach multiple languages. The search service, which is the core feature of the application, is a unified interface to search entries from different dictionary resources in multiple languages (over 40).

The technology used previously in Sanakirja.fi search, was rather old and conservative. Data was retrieved from multiple data sources and as a result, the search was sluggish. Since there was data from multiple dictionaries, users had to select the dictionary to search for, limiting the usefulness of the whole system.

As a part of this project, one goal was to form a unified query and storage subsystem. The new system had to be flexible, allowing users to look for entries in multiple dictionaries and data resources without choosing. It had to be highly efficient in performance on desktop computers and even on low powered mobile devices. It should also provide a functionality of searching for phrases and idioms.

Navigating through all the debris information from different resources

also required an efficient ranking system, so that, the user can only view the information he needs to view, rather than loading him with too much information. Therefore, ranking was another major issue to resolve in the new system. To solve the ranking problem, we combined a variety of methods used in information retrieval. This includes, finding textual weights, tracking user preferences for dictionaries and learning to rank on user feedback.

In the end we try to learn the relevance feedback marked by the users for the documents retrieved by each query. We record the user feedback and use it as a training data for the classifier. The classifier learns from the coefficients for features we use and we use these coefficients combined with the string distances to improve the ranking of the system.

1.2 Structure of the Thesis

The thesis is divided into the following sections:

Chapter 2 is a generalized overview of structure of electronic dictionaries and how to store them. It then describes, the traditional models used in information retrieval. We list down the advantages and disadvantages of these models. In the last section, we describe the learning to rank principle and the different approaches, used for learning to rank.

Chapter 3 gives a brief overview of the search service system and subsystems interlinked to it. It also describes the file format, we are using and how does the import process takes place.

Chapter 4, describes the methods we have used in the search service. It starts with the description of the database structure. In the next section, we talk about the different types of indexes. The following sections talk about preprocessing steps for information retrieval process and various methods used to measure similarities between strings, used for ranking. The last two sections of this Chapter, describe the techniques of Logistic Regression and Cross Validation.

Chapter 5 gives an overview of the implementation of the methods described in Chapter 4. It starts with the database structure and the schema that we have used to generate the database. Later, it talks about how we have implemented indexing in the database. This chapter also elaborates on the details of different types of searches, implemented in the system and the steps that lead towards ranking. In the last section, we introduce the Classifier, that we use for validating the documents and ranking them. It also talks about the dictionary preferable search for users.

Chapter 6, evaluates the different methods that we use in Chapter 5. Since, we have implemented multiple methods for searching and ranking, it

talks about various methods for evaluation. This Chapter starts by introducing methods to evaluate an information retrieval problem. It follows by evaluating performance of different indexes and later in the chapter, we evaluate the Classifier. In the end we compare the rankings for Approximate search, with the previous ranking system used by Sanakirja.fi.

Finally, Chapter 7 concludes the thesis by summarizing the main results obtained.

Chapter 2

Background

When starting to develop the search system, we had a vague understanding of the issues that were needed to be solved by the new system and what new features can be implemented. These requirements were provided largely by the users, using the previous system and research done by Kielikone staff. Other features, used in different electronic dictionaries provided additional inspiration and motivation. For an electronic dictionary to be accurate, the process of revising and restructuring lexical data never ends.

An important aim of this thesis was to develop a search service which is flexible to all different variations of linguistic data. The system should provide support for different dictionaries under one unified interface. We related this problem as a subset of information retrieval, where you can search through all kinds of data and still get the thing you are really looking for. All of the search engine uses these approaches. Before starting this project, we had experimented with full text search. It was returning too much noise in the data, which motivated us to have an efficient ranking system that resolves the noise and returns the user what he is searching for. We tried to use an incremental approach developing feature by feature and an iterative development process just to be sure, that we come up with things what the user is really looking for. Despite some significant improvements in electronic dictionaries, the user is still returned with information, that he is not looking for and it confuses him as someone who is trying to learn a new language.

The first section of this chapter gives an overview of the electronic dictionaries, their structure and storage. This is followed by the background about information retrieval and different types of information retrieval models. The last section is an overview of the different types of learning to rank approaches and their usage examples. This chapter also provide the advantages and disadvantages of the different approaches used.

2.1 Electronic dictionaries

An electronic dictionary is a way of representing the traditional printed dictionaries, with more powerful search functionalities. The content to search can be very large, however, by using optimization, we can also provide additional features including multimedia content such as sound, videos and images. The content storage databases are normally very extensive which contains up to some million headwords and definitions, synonyms, inflected words etc. With the functionality of lemmatization and stemming, multiple words can be indexed for a headword in the base form. When using electronic dictionaries, the user tends to make bilingual searches. The user has limited knowledge about the language he is searching for. The only thing he knows is the keyword, phrase in a particular language and he needs to know what are the translations, examples and definitions in the other language.

2.1.1 Structure and types of electronic dictionaries

The main component of a dictionary is the headword entries or the keywords. Each record or an entry, from the printed dictionary is indexed in the system, on the basis of the main headwords. Other components include the senses, definitions, examples and translations associated with that headword. Electronic dictionaries allow, at minimum, searching for the headword and retrieving all the information associated with that headword. The headwords are stored in a random order and retrieved on the basis of what they are searched for. There are additional fields related to these headwords called the indexes, which tells the system, when to retrieve the headword. For instance, there can be multiple indexes for a particular headword. An example is of headword 'Cats and dogs', which can have both 'cat' and 'dog' indexed. Hence the headword is retrieved for both cat and dog. We should always index the headword in its base form so that the inflected words are also retrieved when the user searches for the base form. Figure 2.1 shows a sample entry search from Sanakirja.fi

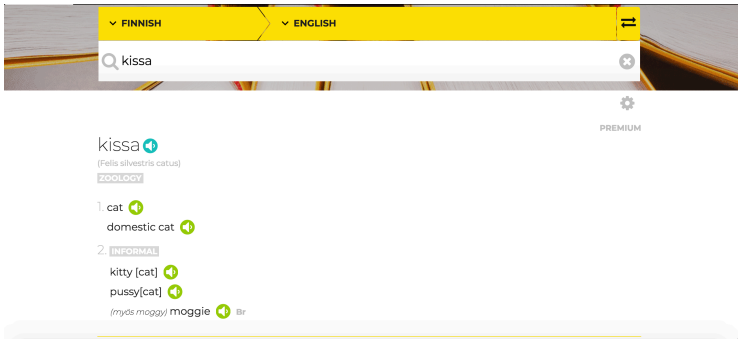


Figure 2.1: Sample dictionary entry from Sanakirja.fi

Element	Example	Explanation
main headword	**peaceful**	first keyword in an entry
index	**peaceful**	the field for which the entry is indexed
phrases	**die peacefully**	phrase containing the keyword
senses		different senses for the keyword
sense number	**1**	identifier for senses
usage examples		example of how the keyword is used
part of speech	[adjective]	which class of word the entry belongs to
translation	**rauhallinen**	translation of the keyword
domain		domain of use
geography	Br	area of use
example	**Tämä on rauhallinen kaupunki**	usage example of the keyword
definition		definition for the keyword
notes	[”adv. peacefully”]	additional information to the headword

Table 2.1: Types of information in dictionary records

The number of records in an electronic dictionary varies from a few hundred to a few million records. There are some additional fields associated with the headwords, which tells us in which context is the headword used. For example, 'geography', 'notes', 'grammar', etc. This reduces the ambiguity of the headword, which changes with context. Table 2.1 shows what type of information is stored with the headword in the records of dictionaries. There are multiple types of electronic dictionaries, which are classified according to their intended audiences and content:

Monolingual dictionaries are dictionaries, which contain the headword and detailed definitions in one language.

Bilingual dictionaries are dictionaries, which contain the keyword in one language and the translation and explanation in the other language.

Multilingual dictionaries have keywords in more than two languages. It is converted from one language to other multiple languages.

Thesauri contains keywords in a single language and words related to that keyword eg synonyms and antonyms

2.1.2 Storing electronic dictionaries

Before the conversion, the printed version of the dictionary is kept in mind. However, there is no standard documentation for storing electronic dictionaries which makes it a manual, rigorous problem, understanding the needs of the system and what information needs to be stored. Many lexicographical and linguistic rules needs to be kept in consideration, when storing these electronic dictionaries. There are separate set of rules for different languages, which are kept in mind and with the help of printed dictionaries, the entries are stored in the electronic form.

2.2 Information Retrieval

Information retrieval is a domain of retrieving documents containing information, relevant to what the user needs to search, from a large collection of documents. The user expresses his needs in terms of a query over an interface, which tells the system what he is searching for. Consequently, the system presents a set of resulting documents to the user. According to [20], information retrieval is to find documents of unstructured nature, satisfying the information needs within large collection of documents. I.R can can also be used to deal with semi-structured data, for example considering a

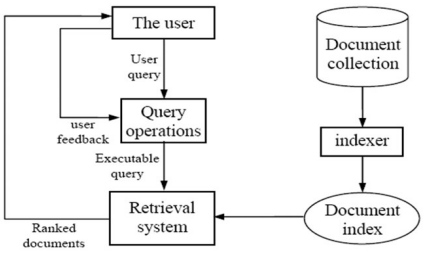


Figure 2.2: Architecture of Information Retrieval System

document, in which, the title contains 'information retrieval' and the body contains 'text recognition'. The field of information retrieval also supports clustering and classification of documents. Clustering is a task where similar type documents are grouped together, based on what content, they have. Classification is a task of grouping, documents into different categories and labels and deciding which one do they belong to.

Basic information retrieval systems, use an optimistic approach. The set of documents returned by the system, using some information retrieval model, are presented in the correct order to the user. On the contrary, other information retrieval systems ranks these results using user preferences. The top ranked results are shown to the user [1, 9]. The database stores information for the documents and the user queries, retrieve documents from the data store. The retrieved objects, can be textual information or multimedia objects like images and videos, depending on the application and user needs.

The information retrieval system takes in information, transforms into a searchable format and provides an interface to allow users to search and retrieve information. The first name that comes to mind, when we talk about information retrieval systems is GOOGLE. The general objective of

IRS should be to reduce the time taken for users to find the information they need. As for a user, time is an important factor which it uses to engage with the IRS system. It also decides the effectiveness of the system [13]. The architecture of information retrieval system is described in Figure 2.2. The user sends a query to the retrieval system, which fetches the documents from the collection, and returns the ranked documents to the user. In return, the user provides some feedback about the documents, whether the document he is looking for, is in the returned document list or not. The retrieval system uses this feedback to improve the document retrieval and ranking.

2.2.1 Information Retrieval Models

Adhoc retrieval is simply retrieving the documents which matches the user query, without measuring the relevance of these documents to the query. Therefore user is bound to look through a large set of unsorted documents to find what they are actually searching for. The user has to interact with the system repeatedly, in order to obtain satisfactory results.

However, in a relevance based retrieval system, there are some measures, that calculates the relevance of the query with the document and retrieve the most relevant documents.

2.2.1.1 Boolean Retrieval Model

This is the simplest form of retrieval [9], where the documents are represented by binary values of 1s and 0s. In contrast to the VSM, in BRM, the users use boolean operators to build up query expressions. This indicates that the query is either present or not present in the document. The query in a boolean model is combined by boolean operators, using AND, NOT and OR. Boolean AND states that both the terms in the query should be satisfied. For example, a query, "cat AND dog" will retrieve all the documents which have both cat and dog in them. Boolean OR is a union of the terms in the query, which means that either of the two terms can be present in the documents retrieved by the information retrieval system. For e.g (cat OR dog) will retrieve all the documents which have either cat or dog in them. Likewise, Boolean NOT will get all the documents which does not have the query terms.

For example, if we have three documents:

- Doc1: Information retrieval and information
- Doc2: Boolean retrieval model

- Doc3: Processed data is information

Then we have 3 terms in the query; Information, Boolean and Retrieval as shown in Table 2.2.1.1. Suppose, we have a query "Boolean OR Retrieval", it will return Doc1 as both of the terms are present in Doc1

Terms	Doc1	Doc2	Doc3
Information	1	0	1
Boolean	0	1	0
Retrieval	1	1	0

The advantages of Boolean model includes:

- Easy to implement
- Easy to understand

The disadvantages of Boolean model include

- At some time may retrieve a few documents
- Only performs exact match
- Does not get the closely related documents

Boolean retrieval model have been the main or only search model for many of the large commercial information retrieval systems, before the early 1990's (the arrival of World Wide Web). There have also been extended Boolean retrieval models in the past as the unordered results returned by the strict Boolean retrieval model are too limited for information needs, that many people have. The extended Boolean retrieval models, measure how close are the two terms in the document by using additional operators called term proximity operators. An IR system implemented on extended Boolean models is defined by the quadruple (T,Q,D,F) [15] where;

- T is a set of index term, representing queries and documents
- Q is a set of queries in the Boolean retrieval system where $q \in Q$ should contain the indexed terms and the logical operators like AND, OR, NOT.
- F is the ranking function $F : D \times Q \rightarrow [0, 1]$ that assigns a weight of similarity to each (q, d) pair ranging between [0,1]
- D is a set of documents

2.2.1.2 Vector space model

A document dj and query qk can be expressed as vectors

$$\vec{d_j} = (t_{1_j}; t_{2_j}; t_{3_j}; :::; t_{N_j})$$
$$\vec{q_k} = (t_{1_k}; t_{2_k}; t_{3_k}; :::; t_{N_k})$$

of t features representing the N terms in the document collection. Each component in a vector is a particular term in the document. The value assigned to that component is the importance or weight of that term in the document. The simplest values in these components can have are binary values of 1's and 0's, which signifies, if the term in the query is present or not present in the document [11].

This binary vector model fails to signify, the importance of each term. As a result, weights are associated, with each term, claiming the importance of each term in a particular document.

In this case, vectors are expressed as weights.

$$\vec{d_j} = (w_{1_j}; w_{2_j}; w_{3_j}; ::: w_{n_j})$$

$$\vec{q_k} = (w_{1_k}; w_{2_k}; w_{3_k}; ::: w_{n_k})$$

where a weight wi_j is the weight of term i in document j. The documents in a collection are represented by a term-document matrix. The columns of the matrix represents the documents and the rows, representing the terms. The values of each element inside the matrix is the weighted frequency of the term i occurring in document j. The relevance is measured by summing up the common terms in the query and document. For eg if we have 3 documents,

- Doc1: The police catches a student
- Doc2: A student meets his friend
- Doc3: A lecturer teaches a student

This can be represented by a matrix:

Terms	Doc1	Doc2	Doc3
A	1	1	2
catches	1	0	0
his	0	1	0
lecturer	0	0	1
meets	0	1	0
police	1	0	0
teaches	0	0	1

Equation 2.1 shows that the similarity between the query and document can be found in vector space by calculating their dot product.

$$sim(d_j, q_k) = d_j \times q_k = \sum_{i=1}^{n} w_i, j \times w_i, k \tag{2.1}$$

In VSM, term frequency refers to the number of times each term has appeared in the document. A numerical weight is assigned to each term, in order to measure the relative importance of these terms in reference to the rest of document collection. Term frequency can be calculated by [1, 2]

$$tf_{i,j} = \frac{C_{i,j}}{\sum_{k \in d_j} C_{k,j}} \tag{2.2}$$

where $C_{i,j}$ are the count of occurrences of terms in the document. Normalization $\sum_{i=1}^{N} w_i^2$ is used to reduce the dominance of longer documents which can have higher similarity.

Another factor affecting the term weighting is the number of documents in which the term appears. This is known as the document frequency or d_f. Many common terms can appear in a lot of documents. Hence, the weightings of document terms can be affected by this. In order to normalize the affect of common terms on the document weightings, we take the inverse document frequency or idf

$$idf = \log \frac{|D|}{|d_j : t_i \in d_j|]} \tag{2.3}$$

$tf \times idf$ gets the standard term weighting score for VSM. It provides accurate measures to the importance of a term in a document, in relation to its importance in the whole document collection [1]

2.2.1.3 Probabilistic Retrieval Model

Given a document d and query q, we take R (d,q) as a random variable which tells us whether d is relevant to q or not. This random variable gets assigned value 1, if the document is relevant to the query and 0 otherwise [20]. In the probabilistic model, the documents are presented to the user in the order of their relevance probabilities as shown in Equation 2.4. This is known as the **Probability Ranking Principle**. The results retrieved by the system are in decreasing order of their probabilities of relevance.

$$P(R = 1|d, q) \tag{2.4}$$

Binary independence Model (BIM) is used with PRP. In this model, the documents and queries are represented as binary term incidence vectors

$$x = (x_1, ..., x_M) \tag{2.5}$$

where $X_t = 1$ if term t occurs in d and $X_t = 0$ otherwise. The term independence here signifies, that the query terms are not associated with each other.

In modern full text searches, the two most essential features, the model should consider are the document frequency and the length of document. **BM25**

$$score(Q, D) = \sum_{j=1}^{n} IDF(q_i) \frac{f(q_i, D) \times (k_i + 1)}{f(q_i, D) + K_1 \times (1 - b + b \times \frac{\|D\|}{\text{avgdl}})} \tag{2.6}$$

where $f(q_i, D)$ is qi's term frequency in the document D, $\|D\|$ is the length of the document D in words, and avgdl is average document length in text collection from which documents are drawn. K1 and b are free parameter $K_1 \in [1.2; 2.0]$ and $b \in [0, 1.0]$. IDF is defined by:

$$IDF = \frac{N - n(q_i) + 0 : 5}{n(qi) + 0.5} \tag{2.7}$$

BM25 has two parameters which are used to tune the results, parameter K_1 controls how quickly an increase in term-frequency should result in its saturation and parameter b controls the field-length normalization.

2.3 Learning to Rank

In information retrieval, learning to rank is a methodology where the system, learns from training data and sorts new entries according to their degree of relevance. In a broader view, LTR means to use machine learning for ranking. There are many ways in which the information system, ranks the results, returned. They work in some cases, however fail in others. To tackle this problem, we use machine learning algorithms to build effective learning algorithms [19].

Learning to rank is a supervised learning task, hence it requires training and testing data [17]. The training data is collected in many different ways. One way is to ask users to explicitly provide relevance feedback for the retrieved entries. An alternative approach can be to use implicit data, by logging the user clicks.

The training data consists of queries and documents. Each query is represented by a unique id and the list of documents retrieved by the query are represented as the associative sets of that particular query. The relevance of the document w.r.t that query is also retrieved. Li [16] uses the approach, where the document relevance is represented by labels/scores at different levels. The higher, the level of the label, the more relevant, the document. Some other methods assigns relevancy scores to each document, retrieved by the query. The higher the score the more relevant, the document is.

If Q is a query set and D is a document set, where $\{q_1, q_2, q_3, ...q_n\}$ are set of queries and q_i is the i-th query. $D_i = \{d_{i,1}\,, d_{i,2}\,, ..., d_{i,n}\}$ is the set of documents associated with q_i and $y_i = \{y_{i,1}\,, y_{i,2}\,, ..., y_{i,n}\}$ is a set of labels associated with q_i. n_i is the number of documents in D and $d_{i,j}$ and $y_{i,j}$ represents the j-th document and j-th label respectively. The original training set is denoted by

$$S = \{(q_i, D_i), y_i\}_{i=1}^{m} \tag{2.8}$$

A feature vector $x_{i,j} = \phi(q_i, d_{i,j}\,)$ is created from each document and query pair, where ϕ represents the features associated with the query document pair. For example, TFIDF weight, cosine similarity, edit distance can be used as features. The main aim of the ranking model is to learn a function $f(q, d) = f(x)$ that can assign score to a given feature vector.

The test data consists of a new query, q_{n+1}, having a set of associated documents D_{n+1}. The trained data is used to assign scores to documents and sort them on basis of the scores given to them.

2.3.1 Approaches to Learning to Rank

The three main approaches to LTR, include pointwise learning, pairwise learning and listwise approach. The pointwise approach and the pair wise approach, transform the ranking problem into, classification, regression and ordinal classification problem. The list wise approach learns the ranking model based on ranking lists. The difference in these models depends in the loss function employed.

2.3.1.1 Point wise approach

The point wise approach, transforms the ranking problem into classification or regression problem. In the pointwise approach, every document has its own feature vector as an input and its own relevance score as an output. The training algorithm, takes a single document as an input and predicts its relevance to the query. It ignores the group structure of the training

data (x_i, y_i) combining, all the groups together. The score of each document is aggregated and added to a resultant list of scores, for the query. Each document is mutually exclusive of the other documents in the resultant list, for that particular query. The final ranked list is formed by sorting the resultant list according to the scores of each document. The loss function is calculated on the basis of how accurately, the hypothesis function predicts the score for each document. Since the documents in the ranked list are not dependent on each other, the position in the list does not affect the loss function.

There are various algorithms for point wise classification and regression.

Polynomial Regression Function

This algorithm, learns the scoring function by using the least square algorithm. A group of documents,

$$x = \{x_j\}_{j=1}^{m}$$

for a query q, the input variable, x_j defined by a vector. For a binary setting, the output variable, y_j is (0, 1), if the document is relevant, else it is (1, 0). If there are multiple categories to judge, y_j becomes a vector, and the k-th element in y_j is set to 1, while all the others are set to 0, given that the k-th category in the document is relevant. The scoring function given by $f = (f_1, f_2, ...)$, predicts the k-th element in y_j. Here, f_k is from the class of polynomial functions

$$\begin{aligned} f_k(x_j) = w_{k,0} + w_{k,1} \times x_{j,1} + ... + w_{k,T} \times x_{j,T} \\ + w_{k,T+1} \times x_{j,1}^2 + w_{k,T+2} \times x_{j,1} \times x_{j,2} + ..., \end{aligned}$$

where $w_{k,l}$ is the combination coefficient, T is number of features and x_j is the l-th feature.

The loss function is defined as the square loss:

$$L(f; x_j, y_j) = \|y_j - f(x_j)\|^2 \tag{2.9}$$

Multi-class Classification for Ranking (Mc Rank)

In a classification setting, all the documents, relevant to a query are classified as positive examples, where as, the non relevant documents are classified as negative examples. The algorithm was proposed by Li et al [18]. Here the authors deal, the ranking problem as a multi class classification problem, taking K (number of classes) = 5. They classify each document into one of

the 5 classes and rank them according to their class labels. The loss function is defined as an upper bound to the classification error. The probabilities of each class is learned from the training data and the ranking function

$$f(x_j) = \sum_{k=0}^{k-1} k \times P(y_j = k) \tag{2.10}$$

Problems with the Pointwise Approach include:

- The documents in the input space are treated as single objects therefore there is no way to predict the relative order of relevance between the documents.
- The position of the document is not available to the loss function and the loss function can give emphasis to the documents lower in the list.
- The loss function can be dominated by the queries with large number of documents as the fact of having similar documents with a particular query is ignored in this approach.

To solve these problems the list wise and pair wise approaches were proposed.

2.3.1.2 Pair wise approach

The pairwise approach consider the pair of documents in the loss function and come up with an optimal ordering for that pair. This approach deals with the problem of learning to rank in a similar way to classification. The learner collects document pairs from the set of documents returned by the system and labels their relative relevance and tries to minimize the miss-classified document pairs. The trainer, uses the classification model with the labeled documents and model the ranking. In the classification model, the relevant documents are classified as +1 where as the non-relevant documents are classified as -1. For instance, if there are two documents X_1 and X_2 and the classifier classifies, X_2 as +1 and X_1 as -1. X_2 will be ranked above X_1. If all the documents pairs are miss classified, the loss function becomes maximum and attains the value of 1 and if all the document pairs are ranked correctly the loss function approaches 0.

Some of then famous algorithms that uses the pairwise approach, includes RankNet, SVM Rank and Rank Boost. L.T.R, using the pairwise approach has been successfully used for information retrieval. Joachims [10] used pair-wise SVM to rank documents pairs derived from users, click through data. Burges [3] applied RankNet to large scale web search. Cao et al. [5] modified the loss function of Ranking SVM for document retrieval.

Ranking SVM

Ranking SVM [10] is a pairwise classification task, which uses the traditional classification SVM. The SVM solution, tries to maximize the margin between two data sets, having different class labels. The ranking model is formed on the basis of minimizing a regularized margin based pairwise loss. Consider a problem for ranking web pages, whose training data comprises of a number of queries, for each query it has a set of documents, a feature vector, for each (query, document) pair and the relevance score of these documents to the query. From this data, we can construct a set of preference pairs P, by comparing the relevance of the documents associated to the particular query. If $(a, b)\ \varepsilon\ P$, then it means that document a has a higher relevance over document b. The objective function, which the Rank SVM minimizes is

$$min\frac{1}{2}\|w\|^2 + C\sum_{(i,j)\in P} l(w^T x_i - w^T x_j) \tag{2.11}$$

where l is the loss function. In this objective function, the margin term $\frac{1}{2}\|w\|^2$ controls the model complexity. The difference between the SVM and Ranking SVM lies in the document pairs constraints. The Ranking SVM loss function defined on document pairs is referred to as hinge loss. For example. for a training query, q if the x_u is more relevant than document x_v and if $w^T x_u$ is larger than $w^T x_v$, by a maximum margin of 1, the loss is 0. Otherwise the loss will be $\xi_{u,v}$ The classifier, judges in pairs that if one document is better than the other one. Instead of working in the space of query document vectors, e.g x_1, x_2 and x_3, we transform the data into a new space, in which the (query, document) pairs are represented by the difference of their feature vectors (w.r.t query) eg $x_1 - x_2$. The output labels, assigned to each of the query, document pairs in the new space are +1 and -1. For e.g if $x_j - x_i$ is assigned with a label of +1, it means that j is more relevant than i. The process is iterated over all the documents for a particular query, and we get the list of ranked documents. SVM performs better than the other proposed methods for supervised ranking [8]. Olivier and Sathiya [6] proposed, a method to optimize the training performance of Ranking SVM using primal Newton method.

It seems that the pairwise approach is a good choice for ranking in most of the use cases. However, it faces challenges which can be even larger than that of Point Wise Approach discussed in Subsection 2.3.1.1. In a pair wise setting, the distributions of pair numbers of each query can be very skewed as the number of pairs formed can be much larger than the number of documents. This problem occurs due to the unbalanced distribution of documents across queries. In this case, the queries with a large number of document

pairs will dominate the loss function and it tends to be inconsistent with the query-level IR evaluation measures. To overcome this problem, Cao et al. [4] and Qin et al. [23] proposed a method to introduce query-level normalization to the pairwise loss function.

2.3.1.3 List wise approach

The list wise approach trains on the entire list of documents and rank them in an appropriate manner. It uses the ranking list for both learning and prediction. As a result, the group structure of ranking is maintained and the loss function incorporates the evaluation measures. It views, the labeled data $(x_{i,1}, y_{i,1})$, ... , (x_{i,n_i}, y_{i,n_i}) as one instance of query. The ranking model is learned from the training data that assign scores to feature vectors and ranks the documents according to the feature vector scores. The feature vectors with the higher scores are ranked higher.

Some of the famous algorithms using list wise approach includes, ListNet, AdaRank and SVM MAP. Cao et al. [5] proposes to use the Luce-Plackett model to calculate the top k probability of list of objects. ListNet employs a neural network model and K-L divergence as a loss function. K-L divergence measures difference between the true ranking list and the learned ranking list, using the top k probability distributions. It uses Gradient Descent as an optimization algorithm. The issue with ListNet is computational complexity. The testing complexity of ListNet is same as that of pointwise or pairwise approach, however the training complexity is in the exponential power of m as an additional m-factorial terms are required to evaluate K-L divergence loss for each query.

AdaRank [29] belongs to the group of list wise ranking methods which directly optimize the evaluation measures. It does that by using a Boosting technique. It creates weak ranker by repeatedly traversing through the re-weighted training data and in the end, combines the weak rankers for prediction. AdaRank is a very efficient and easy to implement algorithm for learning to rank.

SVM MAP [30] is an algorithm which can find a globally optimized solution to minimize the upper bound of the loss function and performs direct optimization of evaluation measures. The key idea behind this algorithm is to consider different upper bounds of loss functions and apply SVM to optimize the upper bounds.

The issue in the pairwise approach was that the trained model is biased towards queries with more document pairs. This issue does not exist in the listwise approach as the loss function is defined for each query. [5]. The listwise loss function can properly represent loss functions whereas, the piar-

wise loss function gives a loose approximation of performance measures. One more advantage for listwise approach is that the listwise loss function converges faster than the pairwise loss function. This means that the RankNet algorithm, using the pairwise approach needs more iterations for training than ListNet.

Chapter 3

System Overview

This chapter contains the short overview of the whole Sanakirja.fi retrieval system. Our focus is on the search service and its relations with other subsystems. We have tried to summarize the functionalities of each system, focusing on their main tasks. The details for these methods are discussed in the later chapters. It also talks about the data file format we are using, complementing with an example.

3.1 System Architecture

The architecture of the system is illustrated in Figure 3.1. The initial file content is in XML or extended XML format. The conversion script takes old XML file data and converts it into json format. The json files serve as an input to the database import script which import the data files into prescribed tables in the data base. Since, there is alot of data to deal with, and frequent changes in the data, the database import process has to robust and efficient.

The query evaluation system or the search service is the core of the retrieval system. It takes the query from the user as an input, evaluates the query, and generate it into an SQL query to the database. It returns the headwords matched as an output. Since the user enters, the query in the user interface system, it is the responsibility of the user interface subsystem, to provide an appropriate view for the results. The search service, returns the results in an appropriate order to the user interface, so that they can be presented easily.

The query evaluator is coupled with other subsystems including the classifier, ranking system, logging system and the evaluation system. The logging subsystem logs the user searches. The classifier takes the results from the

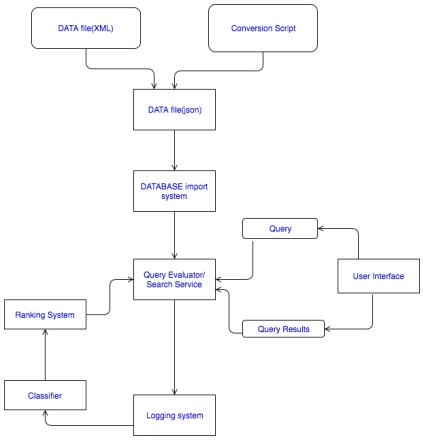


Figure 3.1: System Architecture of Sanakirja.fi search service

logging subsystem and generates the weight coefficients for the ranking system. The ranking subsystem provides results to the query evaluator in a ranked order. The evaluation sub-system, on the other hand, evaluates the performance and accuracy of the ranking and tries to improve them.

3.2 Data file format

Before the entry or record is stored into the database, it is converted into a json format (Douglas Crockford 2000). JSON or JavaScript Object Notation format is a semi structured, file format, which serves as a replacement to XML (Extensible Markup Language) format.

The data file is divided into records or entries which forms the fundamental unit of information for the query system. The main feature of the dictionary record is the headword, which serves as a separate document and contains all the information, related to that headword.

A sample record is shown in Figure 3.2. After the headwords, comes senses in the data hierarchy. The senses are different representations of similar headword, linked by lexical relations. For example a headword ‘horse’ can have multiple senses; one with a translation ‘hevonen’and other with translation ‘hevosvoima’. The senses have translations, examples and definitions linked to them. There are other properties related to the headword

which provide additional information about the headword which includes geography, notes etc.

3.3 Data base import Subsystem

After we have the data in the appropriate format, the data is passed as an input to the database import subsystem. This has a predefined, template for the database schema, which was designed after considering all the possible properties related to the headword and their data types. The import process, takes the json data file as an input, traverses through each record and save it in the related tables in the appropriate format.

3.4 Query Evaluation System

The query evaluation system or the search service forms, the foundation of the search system. It receives, the query from the user interface, transforms it into database readable format and fetches the relevant records from the database. There are three main features of the system: the keyword search, the prefix search (the auto complete) and the search for phrases and idioms. The prefix search returns the list of headwords, on the basis of the prefix match between the record and the headword. The keyword search returns the list of headwords and the properties associated with the headword, fulfilling the query criteria. The user can also query multiple keywords with spaces in between them for example 'cat dog'. This returns results using the wild card search.

Figure 3.3 shows the different type of searches that Sanakirja.fi offers.

3.5 External Components

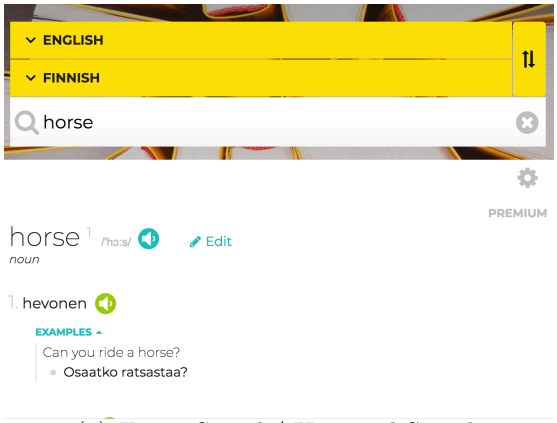
The query evaluation system is coupled with other subsystems including the classifier, ranking system, the logging system and the evaluation system.

Logging System: The logging system logs user specific search queries and the results returned by the query evaluation system. It also logs the user clicks, which forms the basis of user specific rankings and the evaluation system.

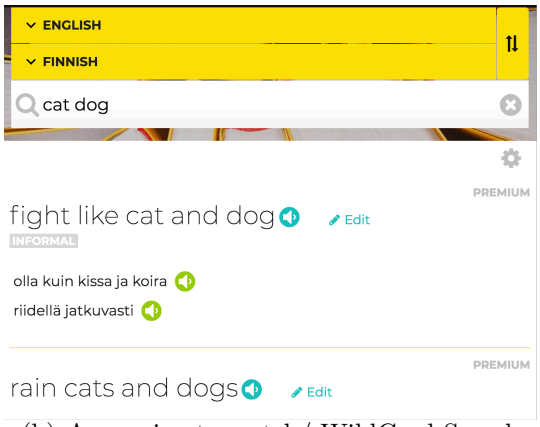
Classifier: It takes in the logged data, with the features and the ranking weights. It takes out the weight coefficients for the important features. The weight coefficients are stored in the database and serves as an input to the ranking system.

```
{
    "word": "interfirm",
    "wordId": "kkenfi-302230",
    "index": [
      "interfirm"
    ],
    "sourceLng": "en",
    "domains": [],
    "headwordvars": [],
    "ipa": [
      {
        "ipa": "ˌɪntəˈfɜːm",
        "note": ""
      }
    ],
    "pos": [
      "adjective"
    ],
    "examples": [],
    "homonym": 0,
    "grammar": [],
    "inflections": [],
    "gender": "",
    "geography": [],
    "notes": [],
    "senses": [
      {
        "translations": [
          {
            "translation": "yritysten välinen",
            "destLng": "fi",
            "trNum": 0
          }
        ],
        "examples": [],
        "definitions" : [],
        "snum": 1,
        "senseId": "kkenfi-302230-s-1"
      }
    ]
}
```

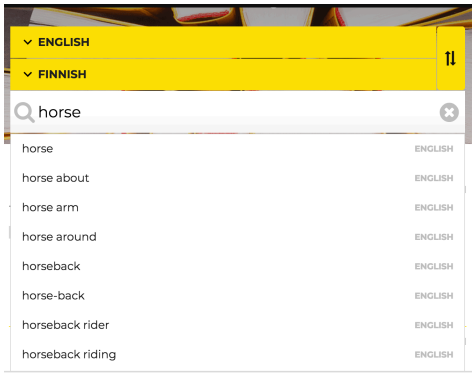

Figure 3.2: Data file Format



(a) Exact Search/ Keyword Search



(b) Approximate match/ WildCard Search



(c) Auto complete/ Prefix Match

Figure 3.3: Multiple searches in Sanakirja.fi

Ranking System: It takes the results from the query evaluation system in a random order and ranks them in an appropriate order. The ranking criteria depends on some relational ordering amongst the headword senses and the string distances between the query and documents returned. There is also user specific ranking, depending on user preferences of dictionaries. We will discuss this in detail in section 5.5.

Evaluation System: The evaluation system supplements the logging system and evaluates the performance and efficiency of the search service. It uses these results to improve the ranking and performance issues.

User Interface: This subsystem is responsible to let users perform searches and display the search results graphically.

Chapter 4

Methods

This chapter talks about the different methods, used to build the search service. The first section, talks about indexing and the different methods of indexing. The next section describes the preprocessing methods used for the retrieving information. It later describes methods used for ranking textual strings. The last two sections describes the techniques for logistic regression and cross validation and its types.

4.1 Indexing

The following subsections talks about different methods that are offered by the database to index records.

4.1.1 B tree indexes

A btree (Comer 1979) is a data structure, that sorts the data, allowing searching, insertion, deletion in logarithmic time. They are preferred, when the data is on the disk rather than the RAM as it takes alot of time to access data from the hard drive. A Btree stores records using nodes with multiple branches called children. When a record is inserted or removed, the number of branches for each node changes. It automatically reorganizes itself and does not need re balancing frequently. The root node is the first node where as leaf nodes are nodes which donot have any further branches. The order of the tree is the number of keys per node.

The internal nodes of the tree is linked with keys. For example, if an internal node has 3 child nodes, then it should have 2 keys k1 and k2. The child nodes on the leftmost must be less than k1, the nodes in between should have values between k1 and k2 and the rightmost nodes must have values

greater that k2. This makes the sorting easy and efficient. The number of keys lies between d and 2d, where d is the minimum number of keys and d+1 is the minimum degree for the tree. The factor of 2 guarantees that the nodes can be split or combined. The number of branches will be one more than the key of that node. For example, if we have d keys, the number of branches of that node will be (d + 1). Figure 4.1 shows a two level b-tree having 3 keys. You can see from the figure that the right outermost node has values < 7, the center node has values between 7 and 16 where as the left outermost has values > 16

B-tree index structure helps to fasten the searches in the database. A simple binary search, in a sorted data structure, containing R records can be done in $log_2 N$. For instance if we have 1 000,000 records, the binary search can be done in 20 comparisons at maximum. Searching in a btree is similar to that of a binary search. The search starts from the root node and traverses from top to bottom. It follows keys belonging to each node and reduces the view of the child node, by only traversing to the part of the tree, which covers the range of the mentioned value. While inserting a new record in the index, the search is made to leaf node and inserts the element in that node, if the maximum number of allowed elements, for that node are not reached. Otherwise, it splits the node and the value, which is in the center of the node is inserted in the parent node. Deleting a record from the btree requires searching for that element in the tree and deleting it from the sub node. Re-balancing of the tree is required, if the number of elements in a particular node becomes less than the minimum number of elements allowed. The re-balancing is done using rotation and merging of the elements.

The advantages of b-tree indexes includes

- the keys are kept in an order so that they can be traversed easily.
- self balances the indexes using a recursive algorithm.
- hierarchical structure is used so that the number of disk reads are minimized.

4.1.2 Trigram Indexes

A trigram of a string, is a 3 element sequence of its characters. For example, if we have a string ‘google’, the trigrams list will be ‘[g, go, gle,goo,le ,ogl,oog]’. It is a variant of n-grams with n=3. Trigrams are often used to find similarity between two texts. Both of the texts are split into 3 character subsequences and while comparing them, we tend to find the number of intersection sets

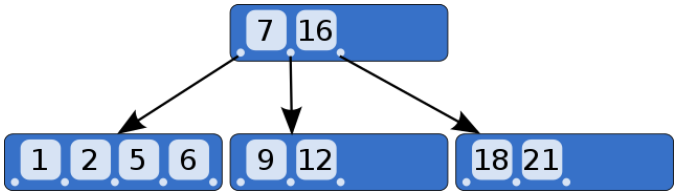


Figure 4.1: A two level B-tree

between the two words. The greater, the number of sequence matches, the higher the similarity between the two words.

Assume that, we have a list of words [‘google’, ‘bottle’, ‘baggage’]. The trigrams for these words will be

Word	Trigrams
google	g, go,gle,goo,le ,ogl,oog
bottle	b, bo,bot,le ,ott,tle,ttl
baggage	b, ba,age,agg,bag,gag,ge ,gga

If we have a query ‘googgles’, having trigrams [g, go,es ,ggl,gle,goo,les,ogg,oog], it has 4 matching sets with the term ‘google’. This indicates that the query term is closer to google than the other two.

4.1.3 GIN

GIN is an abbreviation for Generalized Inverted Index. They are used for cases where, there is a need to have indexes on multiple items and queries, needed to search an element from that multiple indexes. An example can be, to index a set of documents and a query will find words from these set of documents.

GIN index stores a set of (key, posting list) pairs, where the posting list consists of the ids of the documents in which these keys appear. The document id can be present in posting lists for multiple keys, since multiple words can appear in a document. A duplicated key is only stored once, in the list of keys, as GIN uses a compact strategy to store keys. The primary goal to use GIN is to have a high performant full text search. GIN can also be used in fuzzy searches where the query result set is very large and we can set a limit to match the most closely related terms. To retrieve the documents, it searches for all the documents in the posting list for the a particular key.

4.1.4 GIST

GIST, also known as the Generalized Search Tree, is a specialized tree structure that provides functionality of searching through a combination of multiple tree structures including R-Trees and B-Trees etc. Like B-Trees, GIST also contains (key, pointers) however, the keys are not only integers. Instead, they can be user defined classes or some property having a true value for all the items that can be reached, by the pointer linked to that key. The issue with GIST is that, it may give us some false matches in the response. The reason for this is that GIST stores each document by a fixed length signature, which is made by hashing each word into a single bit or taking the union of multiple words. There is a possibility, that two words can have the same hash value and in order to retrieve that word there should be a scan through all the results and check, if they are true or false. This makes searching in GIST a little slower and degrades the performance.[22]

4.2 Retrieving Information

This section includes the basic methods that we need in the preprocessing step for information retrieval.

4.2.1 Tokenization

Tokenization is the process of breaking down data into multiple streams and tokens that can be termed as individual characters. In information retrieval, it is a preprocessing step, which is used to break the query and documents into smaller tokens. For example, if we have a query, ‘cats lives in big houses’. Tokenization, breaks it into cats—lives—dogs—houses

4.2.2 Stemming

Stemming is a process, which finds the root form of the word by removing the morphological and inflectional endings. For e.g run, running, runs and ran can be transformed to the stem word 'run'. Stemming takes place in the preprocessing step and we index the stemmed words in the document.

There are multiple types of stemmers used for english language. However we talk about the two most commonly used.

English Stemmer / Porter Stemmer

English stemmer is update of the Porter stemmer [21]. Porter stemmer, removes the common morphological and inflectional endings from words in English. For e.g the word recommenders stem to recommend

KStemmer

K-stemmer [14] is a morphological analyzer that reduces morphological variants to a root form. For example, 'houses' stemmed to 'house', 'amplification' to 'amplify' and 'italian' to 'italy'. K-stemmer tries to avoid blended variants that have different meanings. For example 'memorial' is related to 'memory', and 'memorize' is also related to 'memory', but reducing those variants to 'memory' would also combine 'memorial' with 'memorize'.

The primary advantage of K-stemmer over Porter stemmer is that it allows any word form to be unified to any other word form. That is, it returns words instead of truncated word forms.

Stop words

Stop words frequently appearing words in the documents and have to be removed before indexing. Doing so, improves the accuracy of the information retrieval system. It also reduces the processing time and the query time as the number of terms indexed, reduces significantly.

4.3 Ranking

After storing and retrieving data, the other main feature of an information retrieval system is ranking the results retrieved by the system. The ranking is based on the relevance of the terms in the query, with the terms in the documents. Since, we are dealing with only textual data, the following

subsections, discuss about ways to rank documents based on string based distances.

4.3.1 Cosine Similarity

Cosine similarity between two strings is a vector-based measure, where each string is transformed to a vector in high dimensional space, such that, strings similar to each other are closer. The angle between the two vectors, which is the cosine angle, defines the similarity of the two strings.
Equation:

$$cos(\theta) = \frac{\sum_{i=1}^{n} X_i Y_i}{\sqrt{\sum_{i=1}^{n} X_i^2}\sqrt{\sum_{i=1}^{n} Y_i^2}} \tag{4.1}$$

is to calculate the cosine angle between two vectors, where X_i and Y_i are components of vector $\vec{X}$ and $\vec{Y}$ respectively.
There are many ways to convert strings into vectors. The most common is the tf.idf which is the dot product of term frequency and inverse document frequency. To keep it simple we are going to talk about these terms w.r.t strings, instead of documents. Term frequency refers to the number of times, each token appears in the string. Whereas, the the inverse document frequency is the inverse of the number of times, the token appears in the strings. This measure is needed to normalize the effect of frequently occurring words or the common words like 'the', 'and', etc.

4.3.2 Edit Distance

Edit distance or the Levenshtein distance [Levenshtein 1965] is another measure of string similarity, which is calculated by the minimum number of insertions, deletions and substitutions required to change one string to another. For example the Levenshtein Distance between 'kitten' and 'fitting' is 3 i.e kitten can be transformed to fitting in minimum, 3 transpositions [24]

1. (substitution of 'f' for 'k')
2. (substitution of 'i' for 'e')
3. (insertion of 'g' at the end).

This similarity is mostly used in correcting spelling errors and finding approximate matches. Damerau (1964) claims that over 80% of the words with spelling errors are most likely to be corrected by a single insertion, deletion, substitution or transposition of two adjacent letters.

The edit distance between two strings of length n and m, can be calculated using dynamic programming in O(nm) time and O(min(m, n)) space. In this process, a $m \times n$ matrix is built, and the edit distance is the minimum cost of comparing the two strings a1..m, b1..n at characters a[i] and b[j]. If we move right in the matrix, it means that we are inserting a character in a, if we move down, we are deleting a character and moving diagonally means that we are substituting or transposing characters. The value at M(m,n) gives the value of the edit distance.

In order to tune and improve the basic edit distance algorithm, weights can be applied to the different edit functions e.g inserting a letter in the text can have a higher weight than deletion or substitution. This metric can fail miserably in some cases so you need to be certain about how to modify it so that your purpose is fulfilled.

Considering the uniformity and simplicity of edit distance, it can be used in almost all different languages. However it has a few drawbacks:

1. It is highly dependent on the positions of the substring in the string as it matches the local similarity

2. Edit distance does not consider the length of the two strings.

4.3.3 JaroWinklerDistance

It is another string metric to find the edit distance between two strings. The lower the JaroWinklerDistance, the more similar, the two strings are. For strings which are exactly similar, the J.W.D is 0, whereas, if the two strings are exactly dissimilar, the JWD is 1.

Jaro algorithm calculates the number of common characters c and number of transpositions t. The mathematical formula for Jaro Distance is calculated by

$$Jaro(x_1, x_2) = \frac{1}{3}\left(\frac{c}{|x_1|} + \frac{c}{|x_2|} + \frac{c-t}{c}\right) \tag{4.2}$$

This algorithm has time and space complexities of O($|x_1| + |x_2|$) [7]

Jaro-Winkler distance algorithm is an improvement to the Jaro Distance, improving similarity measures for initial characters of the string. JWD is equal to the Jaro Distance, plus a score if prefixes of string are equal. The reason for that is less errors are found in the start of the string. Jaro-Winkler Distance is calculated by [7]:

$$JWD(x_1, x_2) = Jaro(x_1, x_2) + (l \times p \times (1 - Jaro(x_1, x_2)) \tag{4.3}$$

where l is the number of characters that agree at the beginning of two strings with a maximum often set to 4. For example 'hamster' and 'hamburger' have l = 3. P is a parameter, usually 0.1, to assign a weight to common prefix. The time complexity fo JWD is also O($|x_1| + |x_2|$) JWD fails to obey the triangle inequality, therefore is not a metric.

4.3.4 Trigram Similarity

Kaplan [12], in his works, observed that having one substring on either side of a word is more efficient than having two or more substrings. His study supports the fact that having trigrams is more efficient than having bi-grams or 4-grams. Trigram similarity is a variant of q-grams method where the size of the string is 3. It is a method of measuring the similarity between two strings. It breaks the strings into trigrams and then measures the similarity between the trigrams of the the two strings.

4.4 Logistic Regression

Logistic regression is a machine learning task, which is modeled on top of the logistic function [26]. Figure 4.3 shows an example of the sigmoid function or the logistic function. In logistic regression, the feature vector X is linearly combined with coefficients to predict output values Y [25]. Equation 4.4 shows an example equation for logistic regression, where y is the output value, b0 is a bias and b1 is a coefficient on x (which is a single feature value). Although, the equation is similar to that of linear regression [28], the main difference is that, the logistic regressor outputs discrete values rather than continuous values. Therefore the output vector Y contains binary values (0's and 1's) for a single class problem.

$$y = e^{(b0+b1\times x)}/(1 + e^{(b0+b1\times x)}) \tag{4.4}$$

This makes it a classification algorithm. The output of the logistic regression hypothesis always lies between 0 and 1. The estimated probability for a logistic regression is

$$P = (y = 1|x : \theta)$$

The logistic regression predicts values y = 1 for probabilities p >= 0.5 and y = 0 for $p < 0.5$. This means for all non negative values, it guesses prediction as 1, and 0 otherwise. The **decision boundary** of a logistic regression function is a single point if x is one dimensional, a linear line between a point, if x is 2 dimensional etc. Logistic regression also tells us about the probabilities which are attained by the two classes.

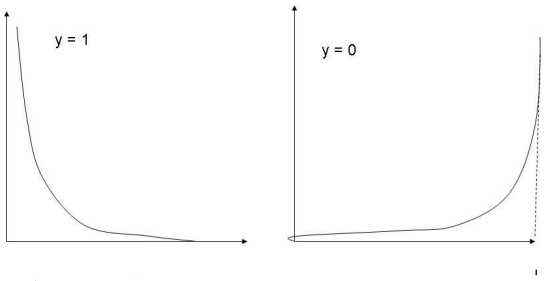


Figure 4.2: Logistic Regression Cost Functions

On the x axis is the

$$h_\theta(x)$$

while on y axis is the cost

The cost function for logistic regression is

$$COST(h_\theta, y) = \begin{cases} -\log(h_\theta(x)), & \text{if } y = 1 \\ -\log(1 - h_\theta(x)), & \text{if } y = 0 \end{cases} \tag{4.5}$$

Figure 4.2 shows the plot of the cost function against the hypothesis function for logistic regression. For Y = 1, the Cost is 0. If h(x) is 1 and y = 1 i.e the model has predicted exactly, what is there in the real data. However, when the hypothesis approaches to 0 the cost approaches to infinity. The second plot shows the curve for Y = 0. We can see from the graph that the cost function approaches to infinity if Y = 0 and our hypothesis predicts values closer to 1. A general intuition from these graphs can be that in logistic regression, we are penalizing the learning algorithm by a very large cost. As a result, we have to find a way to minimize the cost function. To minimize the cost function, we can use gradient descent [27], that is to repeatedly update the parameters using the learning rate or can use some advance optimization algorithms like conjugate descent.

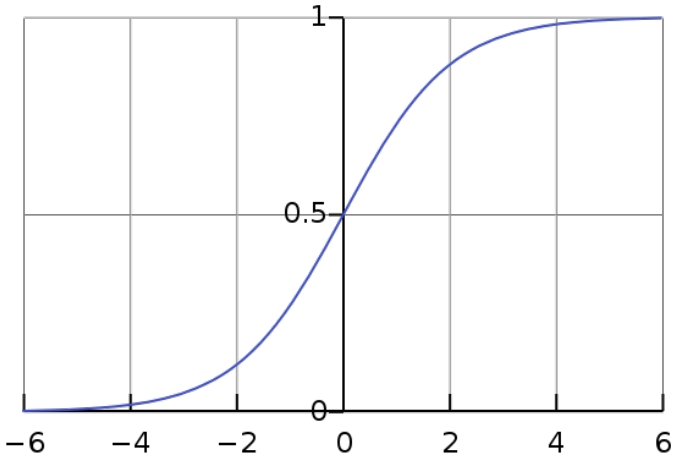


Figure 4.3: Sigmoid Function/ Logistic Function

Sigmoid function is a function, for which the y axis ranges from 0-1 for all values of the x axis. As the values on the x axis approaches to positive infinity, the y axis values approaches to 1. On the other hand, the values of Y axis approaches to 0 for negative values in the x axis going approaching to infinity.

4.4.1 Regularization

Regularization is a technique that resolves overfitting in data. **OverFitting** occurs when our model learns the data to an extent, that it predicts results too well on the training data. However, if we run the same model on unseen data, the accuracy decreases drastically. An example can be considered of a model which has an accuracy of 90% on training dataset. However, if we run the same model on some unseen data or the test dataset, the accuracy decreases to 50%. It can occur if, we have too many features in the training set, the model can fit the training set well, with a cost function of almost 0. However, it does not generalize well enough on unseen data. Here, the term, generalize refers to how well, our model applies to new examples or unseen data. Overfitting, occurs when the model picks up the noise from training samples and learns them as a part of data. The model is too complex to run on unseen data. Another term for an overfitting model can be that the model has high variance.

To avoid overfitting, we use regularization. In regularization, we use all our features, although, we minimize the effect of the features on the cost function. This makes our hypothesis simpler and less prone to overfitting. Mathematically, we add an extra, penalty term to the cost function, λ and multiply it with all the feature coefficients θ. Types of regularization include L1 regularization and L2 regularization.

L1 regularization or Lasso Regression (Least Absolute Shrinkage and Selection Operator) regularizes the cost function by adding an absolute value of the coefficient as a penalty to the cost function. This method reduces the less important features coefficients to zero, removing them from the equation. Therefore it is efficient for feature selection.

L2 regularization or Ridge regression is a regularization technique which adds the squared magnitudes of the coefficients to the cost function. If λ is very large it can add too much weight to the cost function and can make then model underfit. However, if lambda is zero, it nullifies the effect of regularization. Therefore it is important to chose the correct value of λ

4.5 Cross validation and types

In a supervised, machine learning setting, we divide the data to be divided into two sets; the training set and the test set. One way is to manually portion the data into two sets that is assigning 20% of the data to the test set where as 80% to the training set. This can be done multiple times to make sure that the training data is randomly picked and the classifier is less

prone to errors on unseen data. This process is also called 'hold out' method to do cross validation. However, by partitioning the data into different sets we reduce the number of samples which can be used to learn the classifier. To resolve this problem we use k fold cross validation.

4.5.1 KFold CV

K Fold Cross validation is a technique in which all the data is traversed by the validation algorithm. There are several k rounds of cross validation and in one round, the data is partitioned, randomly into subsets being, the training set or the test/validation set. The results are averaged over the rounds. The data is divided into k subsets and if one of the k subsets is used for test set, the other k-1 subsets will be used for training sets. The average error is computed across all the k trials. The advantage of this method is that every data point gets a chance to be in the test set, no matter how the data to be learned is divided. If we increase the value of K the variance in the data is reduced. One disadvantage of this method can be the computational time, the algorithm takes as it has to run k times, every time we run then algorithm.

4.5.2 Leave One Out CV

It is a specific case of KFold CV where K = N, N being the total data points in the sample. In this method, we have only one data point in the validation sample and consider the rest of the data points for training the model. Using this method, we have as many errors as the number of data points in the training sample. To get the final error we take out the sum of errors and divide it by the number of data points.

$$\frac{\sum_{n=1}^{n} E_n}{N}$$

The issue with this validation method is also that it takes alot of computational time, since the algorithm has to iterate through all the data points.

4.5.3 Bootstrap Method

In this method, we randomly draw data sets from the training sample. Each bootstrap sample or the newly created sample has the same size as the original training sample. In this method, we select data points with replacement so the data points can be selected multiple times. We then, refit our model with the bootstrap samples and examine the performance and error of the

model on these bootstrap samples. If the model statistics are consistent across the bootstrap samples, it means that our model has been validated correctly.

Chapter 5

Implementation

This chapter provides a short overview of the implementation of the system and how the different methods discussed in Chapter 4 were implemented. The first two sections describes the database structure and the implementation of the module which generates the database schema. The next section elaborates on the implementation of indexing in the database. This chapter also talks about the implementation of the Approximate search and the Dictionary Specific Search. The last section provides detailed information about the implementation of the Ranking system, describing the different variations for Cosine Similarity and Edit distance. It also talks about the implementation of the logging system and in the end, the classifier.

5.1 Database Structure

Since the data was versatile and large, there had to be an efficient way of storing the data in the database. There was great emphasis on the database structure. Since Postgres DB offers a wide variety of storing data in multiple forms, choosing the right form was a bit of a challenge. We had to keep in consideration, the normalization of the tables, so that the data is not duplicated and redundant. There were separate tables for different entities related to the headword. Figure 5.1 shows the database schema indicating different tables and relations. The headword is the main entity. Next in the hierarchy are senses. There is a $1 \times N$ rel between the headwords and the senses. The definitions, examples and translations are linked to the senses. The indexes table is linked with the headwords as they are different variations of the headwords and are indexed for particular headwords. Each headword is stored with UTF-8 encoding which stores ASCII characters using 1 byte per character and non-ASCII characters which are used for European Languages

in 2 bytes per character.

5.2 Database Generator

The data base generator is a separate micro service which has a template for all the required tables and the schema for the database. The relevant tables are automatically generated using a python script. Since the user retrieves headwords using language pairs, and the data was very large (more than 80 million records), we decided to implement partitioning into the database. The database was partitioned into language pairs and the records for each language pair were stored in the relevant tables. For e.g if there is a word cat – en, which has a translation kissa – fi, there is a separate table for en-fi headword which stores this record.

Another script reads the data from json files and stores records in the relevant tables. There are several properties for a headword which are not just strings, instead they can be objects or list of strings. These fields were stored in json format in the database.

5.3 Indexing

The size of Linguistic materials exceeds the internal RAM memory of the typical computers. The performance of the search is what matters in the information retrieval system. In majority of situations, instant feedback is desired, which should be a few milliseconds. If we try to make linear search with O(n) complexity, the system will take forever to return the records, considering the size of data, we had. Therefore indexing was essential and choosing the right index was equally important.

To be effective, the index has to be externally located on storage medium such as the hard-disk. The reason for this is that, the indices can be very large, having million of records and most of them have to searched at once. With the modern systems, the cost of read and write operations have been reduced significantly making them fast.

As discussed in section 4.1.1 and 4.1.2, we used both BTree indexes and Trigram indexes considering the benefits they have over other indexes and the requirements for the search. The Trigram indexes were used using GIN operators as they are more efficient for retrieving records, as discussed in section 4.1.3.

Since, data had to be gathered from multiple tables, the indexes had to be made on several columns within those tables. The setting for the index

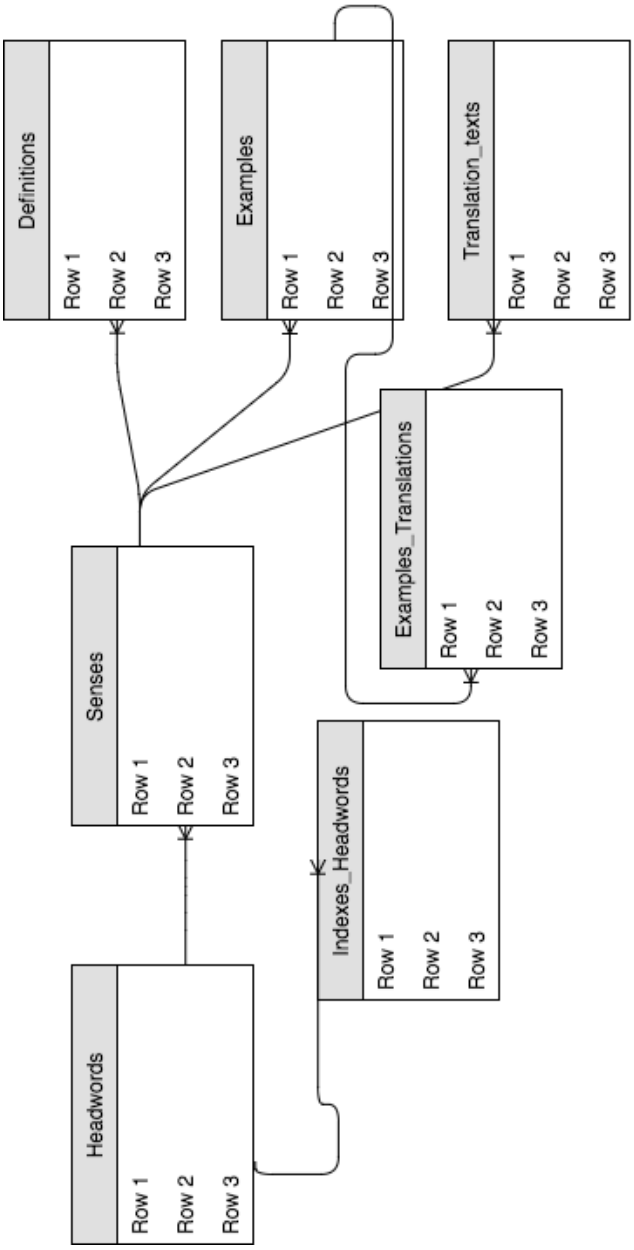


Figure 5.1: Database schema

is as following; The data, as shown in Figure 3.2 has an index field, which signifies the headword to search for. Considering, the fact that this field is a text field and this is the column to search for, we had to make a trigram index on this. Since, this field is a text field, there was an issue of searching for case insensitive words. This was resolved using the trigram indexes, as postgreSQL uses the trigram index for case insensitive searches as well.

The BTree indexes are made on foreign key columns which retrieved data from different tables as they were mostly numerical values.

5.4 Approximate Search

The approximate search was used to retrieve phrases and idioms from the collection. A wild card search algorithm has been implemented to make it happen.

If a user types in a query with multiple keywords, the query evaluation system, tokenizes the query on spaces and forms the wild cards. The wild cards are also formed in the reverse order. For e.g if the user types in the query 'cats dogs', the query evaluation system will break it into two tokens, [cats, dogs] and will form two wild card queries 1)'%cats%dogs' and 2)'%dogs%cats%'. The search system gets results for both of these wild card queries. The advantage for this query is that the user gets some results even if the exact match is not present in the database.

The queries are very fast as postgreSQL uses the trigram indexes for this query. To resolve unwanted results rank them, we have build the ranking subsystem and the relevance evaluation subsystem which are discussed, later in this chapter.

5.5 Dictionary Specific Search

We have headwords stored from a large set of multiple dictionaries including the 'general dictionary', 'technical dictionaries', 'travel dictionaries' etc. In the previous system, the user had the option to select the dictionaries, and the records only contained in that dictionary, were shown to the user. This limitation was however removed, in the current dataset and all of the dictionaries were merged into one storage location. This brought in a problem of ranking the records from multiple dictionaries. The problem is that, if a user searches continuously for records from technical dictionaries, it is most likely that he is interested in technical words. Then, if he searches for a word that appears in more than one dictionary, he should view the technical words first.

id text	email text	count bigint	dictionary text
8afb620b...	naufal.khalid+1991@kielikone.fi	279	tekaen
8afb620b...	naufal.khalid+1991@kielikone.fi	90	motsaksa
8afb620b...	naufal.khalid+1991@kielikone.fi	1261	motmax
8afb620b...	naufal.khalid+1991@kielikone.fi	371	ugc
8afb620b...	naufal.khalid+1991@kielikone.fi	93	tekade
8afb620b...	naufal.khalid+1991@kielikone.fi	137	motespanja
8afb620b...	naufal.khalid+1991@kielikone.fi	279	motenglanti

Figure 5.2: Dictionary count for specific users

A general example can be, if a user has preference for technical words and is looking for the word 'apple'. The results shown to the user will containing both apples, one from the technical dictionary which is the company 'Apple' and then from the general dictionary which is a fruit. Since the user has preference for technical words, he should be presented with technical apple first.

The first step to resolve this issue was to log the user searches, to track which words he is searching for. This way, we had an idea of the words, the user has searched for. Then we had to find the dictionary that words belonged to and count the number of searches the user makes for each query. The later part, required a specialized, SQL query, to fetch records from multiple tables and get the count of dictionaries that each word points to and return the aggregated count for each dictionary requested by the user. Figure 5.2 shows an example result set with the count of number of times, user tries to search for a dictionary. Since motmax is a general dictionary, it means that the user is mostly interested in general terms.

The result set is was transformed into a $1 \times d$ matrix M1, where d is the number of dictionaries, we have in the system. The values of the matrix represented the count of words retrieved, by the particular user from these dictionaries. We also have the result set from the retrieval system containing the headwords, matching to the query. The result set is transformed into a

matrix M2 $d \times n$, where d represents the dictionaries and n represents the words in the result set. The values of this matrix are binary, for instance if the word is present in a dictionary, the value will be 1 else 0. To get the final scores, the two matrices are multiplied into a $1 \times n$ matrix which has the dictionary weight for each headword. This final score was used for ranking.

We tried to implement this algorithm at first but the issue with it was that, it fetched the dictionary count from the database and weights to be recalculated on each query, making it slower. As a result we found out a way to write a cron job which automatically queries the user dictionary count. Based on that count, selects the dictionary, the user has preferred the most and then add that to the user profile. The user dictionary preference is send to the search system with the search request and hence the system knows before hand about the users, dictionary preference. This makes the ranking more efficient in performance.

5.6 Ranking System

We tried several methods to rank the results, phrases in particular, returned by the search system. These methods include:

5.6.1 Cosine Similarity

We had to find the cosine similarity Subsection 4.3.1 between the query and the each of the record returned by the system. As a preprocessing step, we had to vectorize the query and the document returned by the system and then take out the cosine similarity. The following algorithms represents the implementation of cosine similarity in the system.

Algorithm 1 Vectorize Text

```
1: procedure VECTORIZE(text) ▷ Remove punctuation, white spaces and text should be lower case
2:     index ← null
3:     text ← text.split                                  ▷ split the string on spaces
4:     while len(text) ≠ 0 do
5:         index[word]    ▷ add the word as key in the index dictionary and value 1
6:     return index                                                              ▷
```

Algorithm 2 Cosine Similarity

1: **procedure** COSINE($query, docList$) ▷ Cosine Similarity
2: $cosine \leftarrow 0$
3: $vec1 \leftarrow \vec{query}$ ▷ Vectorize query using Algorithm 1
4: **while** $len(docList) \neq 0$ **do**
5: $vec2 \leftarrow \vec{doc}$ ▷ For each doc in document list, vectorize doc using Algorithm 1
6: $cosine = Cosine_Sim(vec1, vec2)$ ▷ Find out the cosine similarity between query and document
7: **return** $cosine$

5.6.2 Normalized Edit Distance

An edit distance as discussed in Subsection 4.3.2 is used to compare the two strings. In our setting, we find the edit distance between the query and the list of documents returned by the system in order to rank the documents. We use the basic version of the edit distance by assigning the same weights to each of the insertion, deletion and substitution operations. The traditional edit distance gives an integer distance and there is no limit to this value. Hence, we had to normalize the edit distances so that the distance stays between intervals $[0, 1]$.

In order to normalize the edit distance, we divide the edit distance between the query and document by the length of the longest word. Violation of the triangle inequality in this distance was not a problem here as we had to find pairwise differences.

$$n.e.d(a, b) = d(a, b) \div max(a, b) \tag{5.1}$$

There were also some other important features used for ranking, including the number of query words in a document and if the query exists in the document or not. The reason for using multiple features, rather than using just one feature was that, one distance worked well for some use cases while, it failed for the others.

After we had these features, we had to build a machine learning system which learns the importance of each feature by assigning weight coefficients to it and then ranking the documents using these weights. We introduce the Classifier in Subsection 5.6.4 but before that we had to log relevance feedback, to collect the training data for it.

5.6.3 Logging user queries

To collect the training set for the classifier 5.6.4, we had to build a logging system, which can track the relevance feedback from the user for each entry returned by the system. The user had to explicitly mark, if the document returned by the query is relevant or not. On the front end, it was a simple button after each document returned by the system. If the user thinks that the entry is relevant, he marks it as relevant. Since it was done as a closed experiment, with the people having knowledge about linguistics, we went with an optimistic approach, that the user marking the relevant documents will go through the list of documents returned by the system and mark each relevant document with the query.

The idea for the logging subsystem came from [10]. Each query had a separate id and a list of documents associated with that. If the document with the query was clicked, the value of 1 was assigned to the clicked field of that document else it was 0. We recorded the user id for each user since each user can mark the relevance separately. The session id was also recorded, since a user can mark a document, associated with a query as relevant in one session and after some time he marks the document as irrelevant. For simplicity, the string distances as described in the previous subsection were also recorded for each query and document.

5.6.4 Classifier

After logging the user queries and tracking all the relevant and non relevant documents retrieved, we built a machine learning system to classify the documents as relevant or non relevant. We use supervised machine learning where the system is trained on a set of predefined training examples and classifies the new input accurately

The data is collected from the query logs table, which records the users feedback for relevant and non relevant documents for each query and the string distances of the keyword and the document retrieved from the system. For some queries, the user has not clicked any of the entries. We filter out such entries. We also track the rank of the document, clicked in the list to get to know how much entries, the user scrolls through, before marking the relevant document. This way we can know how many entries, the user is really interested in and can get the top K documents retrieved instead of returning the whole list.

Table 5.6.4 shows on over view of the table we maintain for the documents and query. The fist five columns contain; query, documents retrieved, the normalized edit distance, cosine similarity, the number of query words

existing in the document and if the query exists in the document or not. The last column is whether, the user clicked on the document or not. This marks whether the document is relevant or not. To remove redundancy in the data, for similar queries and document marked, we take the mode of the click value. For instance, for a query 'catch on', if 7 users have marked the document 'catch on' as relevant and 3 users have marked as irrelevant, then we mark it as relevant for the query.

This data is fed to the logistic regression algorithm. The input feature vector X contains [N.E.D, C.S, WExists, QExists] and the output vector Y contains values for clicked and not clicked.

We divide the data into training at tests sets using cross validation technique as discussed in Section 4.5. Since, the data to train the algorithm was not huge, we manually divided the training set and test set such that we assign 80 % of the data for training and 20 % for test. The classifier, predicts the values for the training data.

The classifier learns on the weights as features and predicts if the document is relevant or not relevant, having those weights. Since the weights are relevant to the query, we do not need to take the query as a separate feature as we use the point wise learning to rank approach as described in Subsection 2.3.1.1.

The logistic regression classifier gives us the coefficient weights for each feature. Since our problem is concerned with ranking the records rather than classifying them, we use the feature weight coefficients and multiply with each of the real value weights of assigned with each document and rank the documents accordingly. For instance, we have a weight vector V1, returned by the logistic regression classifier, which gives the importance of each feature in our system. Then we have another vector, V2 which gives us the real values of string distances assigned to different documents. We take the dot product of V1 and V2 vectors and get the final average weight assigned to each document. Equation 5.2 shows the final equation of cumulative weight assigned to each document where C is the coefficient and f is the real value of feature weight.

$$C = \sum_{n=1}^{t} C_n \times f_n \tag{5.2}$$

Query	Doc	NED	CS	WExists	QExists	Clicked
catch on	catch one's breath	0.55556	0.35355	0.5	1	1
catch on	bayonet catch	0.769	0.50	0.5	0	0
catch on	catch on	0.00	1.00	1	1	1
koira	koira	0.00	1.00	1	1	1
koira	kuin kissa ja koira	0.737	0.50	1	1	0
koira	koiran karvat	0.62	0	0	1	0

Chapter 6

Evaluation

This chapter contains an overview of the evaluation of the search system. The time efficiency, search performance, the improvement in query performance, when using the different indexes. In the later part of this chapter, we discuss about the performance of the classifier. The last section of this chapter compares the ranking of the approximate search in the new system with the old system intuiting that the new system ranks the documents better.

6.1 Precision, Recall and Fscore

Information retrieval system returns a set of documents for a query. Relevant documents retrieved by the user query are called true positives (tp) and those not retrieved are called false negatives (fn). Non relevant documents retrieved by the user query are called false positive(fp) and those not retrieved are referred as true negative(tn). **Precision** is a measure of performance of the retrieval system, which is the fraction of relevant documents retrieved to the number of documents returned to the user.

$$P = \frac{tp}{tp + fp}$$

Recall is the ratio of relevant documents retrieved to the user to the total number of relevant documents in the collection.

$$R = \frac{tp}{tp + fn}$$

	Relevant	Non Relevant
Retrieved	true positives(tp)	false positives(fp)
Not Retrieved	false negatives (fn)	true negatives(tn)

There is a trade of between precision and recall within an information retrieval system. A high precision indicates that the system returns a low proportion of non relevant documents. On the contrary, a high recall score signifies that a large number of relevant documents, available in the collection have been returned by the system to the user. If we increase the number of documents returned by the system, there are more chances of retrieving relevant documents, increasing the recall while decreasing the precision. On the other hand, if we retrieve less documents, the precision will go high where as recall decreases.

If both precision and recall are equally important within a system, we calculate their harmonic mean, known as the F-score

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

6.1.1 Precision at K

For modern, information retrieval systems, calculating the recall does not give us meaning full evaluations as there can be many relevant documents with the query and the user is hardly interested in going through all of them. Instead, precision at top K or P@K can be a useful metric. For example P@5 refers to Precision at 5, which is the number of relevant results returned by the system for the first 5 documents. It ignores all documents with a rank lower than K.

6.1.2 Mean average precision

Mean average precision is for a set of different queries and at different results. MAP provides a single figure performance result for a search technique (one set of setting) over all quires and all precision levels.

The results are retrieved from the relevancy system as discussed in Subsection 5.6.3. We calculate the average precision for the top 5 documents for different queries and then calculate M.A.P. The average precision for query is calculated as a mean of precision at all relevant documents before the k^{th} relevant document, where k is a free parameter. If the set of relevant documents for an information need $q_j \in Q$ is $\{d_1, ...d_{mj}\}$ and R_{jk} is the set of ranked retrieval results from the top result until you get to document d_k, then

$$\text{MAP}(Q) = \frac{1}{|Q|} \sum_{q=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} \text{Precision}(R_{jk}) \tag{6.1}$$

6.2 Comparison between different indexes

This section provides an overview of the the postgreSQL plans. For the consistency of results, we have used the same query. In the experiments, we add b-tree indexes and trigram indexes on different columns and tables of different size. We compare the size of the index and the execution time it takes for postgreSQL to execute the query. We have the headwords table which is the focal point for all the queries. Since the query fetches the headwords from the table, we will try different indexes on the headwords column. We will use prefix queries for our experiments, since it gets a large number of rows. There are approximately 460K headwords in the table, making it large enough to use the indexes. The table below shows a comparison of the query, evaluated without index and evaluated after creating Btree indexes and Trigram indexes. Figure 6.1 shows the postgreSQL query plans for different indexes. From the query plans, trigram indexes outperform btree indexes even though the size is not that much. Since we have case insensitive searches, we also experimented B-tree and trigram indexes with case insensitive searches using 'ILIKE' operator. It turned out that postgreSQL does not use b-tree indexes for case insensitive search, whereas it uses the trigram indexes. Even for phrase or wild card search like '%ab%' postgreSQL query planner does not use btree indexes. Figure 6.2 elaborates the fact that postgreSQL makes a sequential scan if we use b-tree index with ilike queries and an index scan if we use ilike queries with trigram indexes.

	Rows returned	Index Size	Execution Time
No Index	1269		557 ms
Btree Index	1269	15 MB	9.737 ms
Trigram Index	1269	17 MB	5.967 ms

Query = EXPLAIN ANALYZE SELECT * FROM headwords where headword like 'ab%';

6.3 Search Efficiency

The basic functionality of an online dictionary, used for day-to-day use, are listed below

- Searching for a specific keyword and looking for a translation, definition or example for the word. The results returned to the user should be accurate, that is the exact match first and there should not be a perceptible delay when performing this operation.

```
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on en_fi_headwords  (cost=5.68..452.79 rows=168 width=289) (actual time=8.656..9.598 rows=1148 loops=1)
   Filter: (headword ~~ 'ab%'::text)
   Heap Blocks: exact=42
   ->  Bitmap Index Scan on idx_headword_btree  (cost=0.00..5.64 rows=122 width=0) (actual time=8.626..8.626 rows=1148 loops=1)
         Index Cond: ((headword ~>=~ 'ab'::text) AND (headword ~<~ 'ac'::text))
 Planning time: 0.500 ms
 Execution time: 9.737 ms
(7 rows)
```

(a) Using btree indexes

```
                                                                QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on en_fi_headwords  (cost=29.30..631.47 rows=168 width=289) (actual time=3.231..4.954 rows=1148 loops=1)
   Recheck Cond: (headword ~~ 'ab%'::text)
   Rows Removed by Index Recheck: 784
   Heap Blocks: exact=574
   ->  Bitmap Index Scan on idx_headword_trgm  (cost=0.00..29.26 rows=168 width=0) (actual time=3.097..3.097 rows=1932 loops=1)
         Index Cond: (headword ~~ 'ab%'::text)
 Planning time: 1.222 ms
 Execution time: 5.089 ms
(8 rows)
```

(b) Using Trigram Index

```
                                                                QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 Seq Scan on en_fi_headwords  (cost=0.00..14824.10 rows=168 width=289) (actual time=0.156..557.019 rows=1269 loops=1)
   Filter: (headword ~~* 'ab%'::text)
   Rows Removed by Filter: 458099
 Planning time: 0.570 ms
 Execution time: 557.280 ms
(5 rows)
```

(c) Without indexes

Figure 6.1: Query plans for different indexes

```
postgres_staging=# EXPLAIN ANALYZE SELECT * FROM en_fi_headwords where headword ilike 'ab%';
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on en_fi_headwords  (cost=0.00..14824.10 rows=168 width=289) (actual time=0.134..532.160 rows=1269 loops=1)
   Filter: (headword ~~* 'ab%'::text)
   Rows Removed by Filter: 458099
 Planning time: 2.253 ms
 Execution time: 532.316 ms
(5 rows)
```

(a) Btree ilike query

```
postgres_staging=# EXPLAIN ANALYZE SELECT * FROM en_fi_headwords where headword ilike 'ab%';
                                                                     QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on en_fi_headwords  (cost=29.30..631.47 rows=168 width=289) (actual time=6.537..19.982 rows=1269 loops=1)
   Recheck Cond: (headword ~~* 'ab%'::text)
   Rows Removed by Index Recheck: 663
   Heap Blocks: exact=574
   ->  Bitmap Index Scan on idx_headword_trgm  (cost=0.00..29.26 rows=168 width=0) (actual time=6.385..6.385 rows=1932 loops=1)
         Index Cond: (headword ~~* 'ab%'::text)
 Planning time: 9.239 ms
 Execution time: 21.392 ms
```

(b) Trigram ilike query

Figure 6.2: Query plans for different indexes using 'ILIKE' queries

- Searching for the word which starts with the query string. This is helpful for the auto complete feature implemented. It uses the prefix search matching.

Table 6.3 shows the variation of average times for prefix search, with and without cache. Since, the database uses cache, if we consecutively search for the word, we do not have to manage it ourselves. We tried using different queries, using only two letters in the keyword from keywords in Finnish English language (463,000 keywords). e.g 'ca%'. The results show that the mean approximate time for the prefix search without cache is 275 ms and with using cache is 205ms. The use of correct indexes effects the performance and makes it efficient. The size of the data retrieved does not degrade the performance since, we are using indexing and caching at the database level.

Number of keywords	time (ms) without cache	time (ms) with cache
8227	400	200
7000	375	270
6289	205	184
5000	201	193
3098	197	179

6.4 Evaluation of the classifier

For the classifier, we had collected the data to be trained, as discussed in the Subsection 5.6.4. We visualized the data for different features, that we selected for the classifier as shown in Figure 6.3. From the figure, it shows that the users are most likely to click on the exact matches for exact search

and for the approximate search, they are most likely to click on the document, for which the query exists. For example, in Figure 6.3 (b), the document, 'catch lock' is clicked for query 'catch lo'.

Figure 6.4 shows the scatter plots of different features, that we are using. The clicked region is marked with red dots. Figure 6.4 (a) shows the variation of the evaluated documents w.r.t features 'Word Exists' and 'Cosine Similarity'. The major portion of the clicked documents lies in the region (1,1). Since ' Normalized Edit Distance'is opposite to 'Cosine Similarity', Figure 6.4 (b) has the clicked documents region, near (0,1).

After analyzing the data, we pass it to the classifier and train the data collected, using the logistic regression classifier. The feature vector X is of dimensions, (369, 4). We use two methods of cross validation to validate the data:

- Dividing the data set into training and test set i.e the 'Hold Out Method'
  Accuracy on training set: 0.801
  Accuracy on test set 0.817

- 10 Fold cross validation
  Cross validation scores [0.78947368, 0.78947368, 0.67567568, 0.94594595 0.67567568, 0.94594595, 0.78378378, 0.70444444, 0.60333333, 0.83333333]
  Accuracy using cross validation: 0.79 (+/- 0.22)

The table below shows the confusion matrix for the classifier 5.6.4.

	Predicted: NO	Predicted: YES
Actual: NO	(tn) = 58	(fp) = 3
Actual: YES	(fn) = 14	(tn) = 18

The classifier also predicts the coefficient vector for the features used.
Figure 6.5 shows the coefficient vectors returned by the classifier for 20 iterations. The first value represents the Normalized edit distance, the second value represents the cosine similarity, the third value represents the number of words that exists in the query and the fourth represents, if the query exists or not. By looking at these values, the normalized edit distance effects the classifier the most in the negative direction. This means that the change in the value of N.E.D. will shift the classifier towards the negative class. Similarly, the change of value in the Cosine similarity increases the probability of the entry to be clicked. Figure 6.5 also shows that the most important feature to get the document clicked is W.E (number of query words existing in the document or headword).

Query	Document	Normalized Edit Dist...	Cosine Similarity	WordExists	Query Exists	clicked
hyvä	hyvää joulua	0.667	0	1	1	0
hyvä	hyvä	0	1	1	1	1
hyvä	hyvähän sinun o...	0.818	0	1	1	0
hyvä	hyvällä tuulella	0.750	0	1	1	0
hyvä	hyvää yötä	0.600	0	1	1	1
hyvä	olla hyväksi jolle...	0.826	0	1	1	0
hyvä	hyvä on	0.429	0.707	1	1	1
hyvä	hyvä puoli	0.600	0.707	1	1	0
hyvä	hyvä alku	0.556	0.707	1	1	0

(a)

Query	Document	Normalized Edit Dist...	Cosine Similarity	WordExists	Query Exists	clicked
catch lo	catch lock	0.200	0.500	0.500	1	1
catch lo	night-flowering ...	0.750	0	0	0	0
catch lo	TAC	1	0	0	0	0

(b)

Query	Document	Normalized Edit Dist...	Cosine Similarity	WordExists	Query Exists	clicked
istua iltaa	istua iltaa	0	1	1	1	1
istua iltaa	suistua raiteiltaan	0.421	0	0	0	0
istua iltaa	kallistua iltaan	0.312	0	0	1	0

(c)

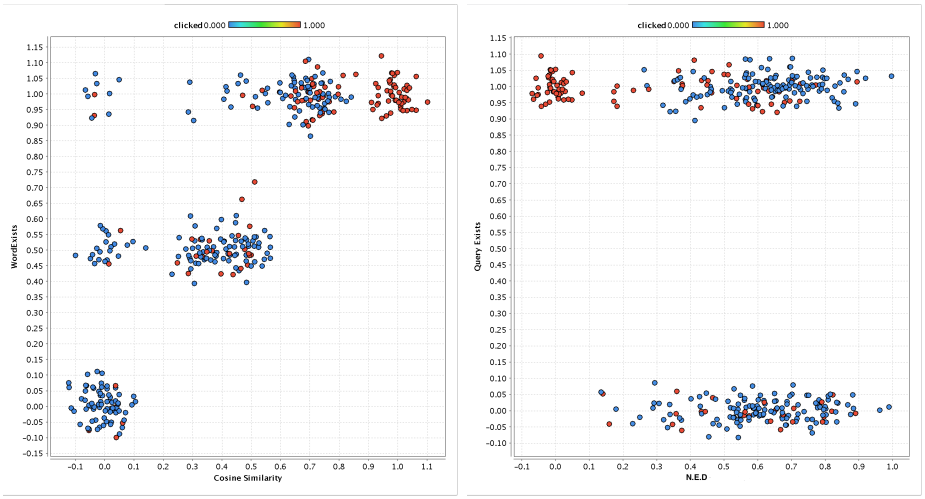
Figure 6.3: Results for different queries

6.5 Approximate search effectiveness

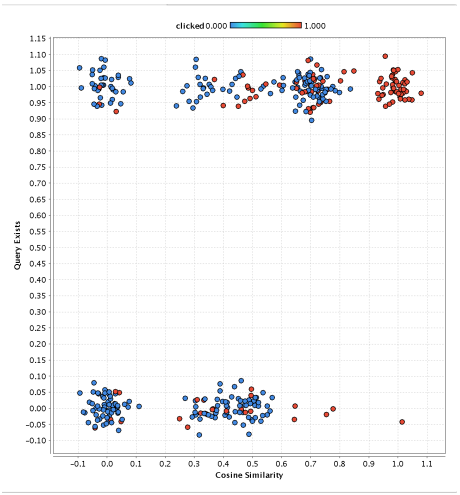
We required a set of typical queries for approximate matching to evaluate the effectiveness of the Approximate Search 5.4 in finding the results. To measure the effectiveness, we use Precision@K as discussed in Subsection 6.1.1. Since we use the coefficient features from the classifier 5.6.4, we perform a comparison of the results with and without the ranking coefficients. The previous Sanakirja.fi system does not use the classifier coefficients for ranking, so we compare the results with that. We perform different queries for approximate matching and calculate the mean average precision at K= 2, K=5 and K=10 for both the systems.

	MAP@2	MAP@5	MAP@10
Sanakirja.fi	0.875	0.6	0.55
New system	1	0.85	0.75

As you can see from the table, the new system outperforms, the existing system in the mean average precision results at all values of K.

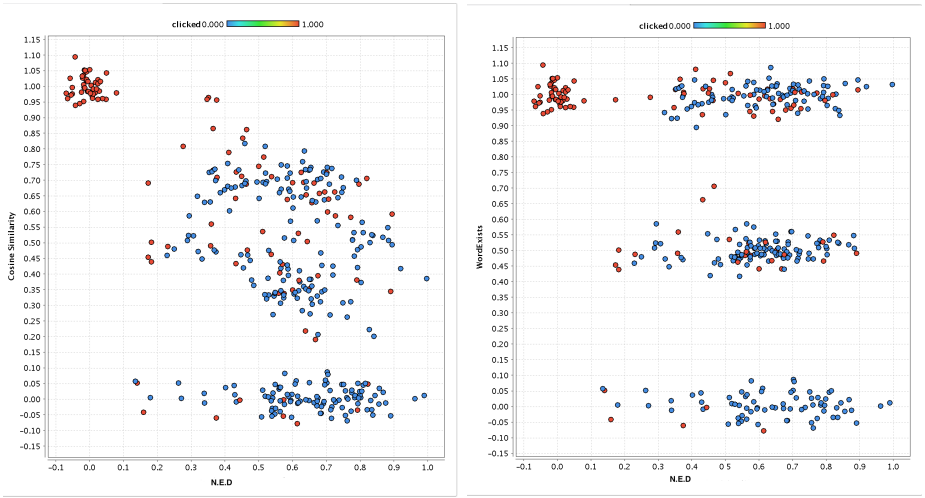


(a) (b)



(c)

Figure 6.4: Visualizing different features



(a) (b)

Figure 6.4: Visualizing different features (cont.)

Row No.	NED	CS	WE	QE
1	-1.183	0.291	0.727	-0.164
2	-1.127	0.708	0.302	-0.086
3	-0.874	0.624	0.150	0.043
4	-0.929	0.995	-0.041	0.011
5	-1.100	0.420	0.554	-0.104
6	-1.107	0.386	0.522	-0.085
7	-1.157	0.650	0.285	-0.148
8	-1.140	0.387	0.578	-0.263
9	-1.122	0.423	0.701	-0.188
10	-1.143	0.459	0.725	-0.364
11	-1.031	0.327	0.663	-0.111
12	-0.964	0.442	0.372	-0.070
13	-1.113	0.559	0.277	0.016
14	-0.961	0.553	0.432	-0.206
15	-0.954	0.454	0.466	0.019
16	-0.905	0.558	0.443	-0.208
17	-0.984	0.644	0.391	-0.067
18	-1.063	0.691	0.280	-0.139
19	-1.202	0.209	0.696	-0.127
20	-1.076	0.260	0.698	-0.113

Figure 6.5: Coefficient Vectors for the classifier for 20 iterations

Chapter 7

Conclusions

Using a single query system and storage system for querying different dictionaries in an electronic dictionary product was found to be both feasible and useful. The system was flexible enough to effectively store and query large amounts of linguistic data and was optimized for lexical applications and linguistic data.

The basic features of the system were keyword search, prefix search, phrase and approximate search. To better serve our audience, the system had an efficient ranking system which measures the closeness of the headwords, retrieved to the query made by the user. The implementation of efficient indexes improved the time needed to store and retrieve linguistic data. The implementation of learning to rank helped to improve the rankings of the documents returned by the system.

The first version of the system has already been integrated in Sanakirja.fi and is successfully used by the public. However, the ranking system is still in testing mode and will be a part of Sanakirja.fi later this year.

In this thesis we try to implement the system using 'standard' Information Retrieval method on linguistic data. There were some limitations since there are set of rules followed by different languages and the data was retrieved from a relational database, rather than in text format. For the ranking system, we had to chose string distances as features, since we were considering the closeness of headwords with the user query. Since the system was relatively new, we had to follow an incremental approach and had to try different things before implementation.

Directions for Future Improvements

The current implementation of the query system is far from complete or perfect. There are still alot of features that need to be implemented like

phonetic search, full text search and near matches search. Currently the data is retrieved from a single data source, however in the future, the system should be able to retrieve data from multiple linguistic sources and merge the results efficiently.

The features used to train the logistic regressor are very minimal. We can add more features to improve the ranking of the system. The data collected to train the logistic classifier was very limited to efficiently train it. Once the feedback and ranking system are moved to production, we will be able to retrieve more feedback from users, hence we can train the classifier better. This feedback will also help us evaluate and analyze, how do different users want to us to rank the documents. Since we had a very few phrases in the system, the approximate search system and the ranking system will be more effective after we implement the full text search. Instead of ranking the documents, we can use the classifier to reduce some of the noise returned by the retrieval system. In the current system we measure the similarity between the query and the headword. In future, we can measure similarity with definitions and examples as well.

Bibliography

[1] Arens, R. J. *Learning to Rank Documents with Support Vector Machines via Active Learning.*

[2] Baeza-Yates, R. A., and Ribeiro-Neto, B. *Modern Information Retrieval.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[3] Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. Learning to rank using gradient descent. In *Proceedings of the 22Nd International Conference on Machine Learning* (New York, NY, USA, 2005), ICML '05, ACM, pp. 89–96.

[4] Cao, Y., Xu, J., Liu, T.-Y., Li, H., Huang, Y., and Hon, H.-W. Adapting ranking svm to document retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2006), SIGIR '06, ACM, pp. 186–193.

[5] Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., and Li, H. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning* (New York, NY, USA, 2007), ICML '07, ACM, pp. 129–136.

[6] Chapelle, O., and Keerthi, S. S. Efficient algorithms for ranking with svms. *Inf. Retr. 13*, 3 (2010), 201–215.

[7] Christen, P. A comparison of personal name matching: Techniques and practical issues. In *Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops* (Washington, DC, USA, 2006), ICDMW '06, IEEE Computer Society, pp. 290–294.

[8] Donmez, P., and Carbonell, J. G. Active sampling for rank learning via optimizing the area under the roc curve. In *Proceedings of the*

31th European Conference on IR Research on Advances in Information Retrieval (Berlin, Heidelberg, 2009), ECIR '09, Springer-Verlag, pp. 78–89.

[9] Fereshteh Mahdavi, and Arash Habibi Lashkari, and Vahid Ghomi, . A boolean model in information retrieval for search engines. *Information Management and Engineering, International Conference on 00* (2009), 385–389.

[10] Joachims, T. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2002), KDD '02, ACM, pp. 133–142.

[11] JURAFSKY, D. Speech and language processing : an introduction to natural language processing. *Computational Linguistics, and Speech Recognition* (2000), 643–644.

[12] Kaplan, A. An experimental study of ambiguity and context. *Mechanical Translation 2* (1955), 39–46.

[13] Kowalski, G. *Information Retrieval Architecture and Algorithms*, 1st ed. Springer-Verlag New York, Inc., New York, NY, USA, 2010.

[14] Krovetz, R. Viewing morphology as an inference process. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 1993), SIGIR '93, ACM, pp. 191–202.

[15] Lee, J. H. Properties of extended boolean models in information retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 1994), SIGIR '94, Springer-Verlag New York, Inc., pp. 182–190.

[16] Li, H. *Learning to Rank for Information Retrieval and Natural Language Processing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2011.

[17] Li, H. A short introduction to learning to rank. 1854–1862.

[18] Li, P., Wu, Q., and Burges, C. J. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in Neural Information Processing Systems 20*, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, Eds. Curran Associates, Inc., 2008, pp. 897–904.

[19] Liu, T.-Y. Learning to rank for information retrieval. *Found. Trends Inf. Retr. 3*, 3 (Mar. 2009), 225–331.

[20] Manning, C. D., Raghavan, P., and Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[21] Porter, M. An algorithm for suffix stripping. *Program 40*, 3 (2006), 211–218.

[22] PostgreSQL.

[23] Qin, T., yan Liu, T., feng Tsai, M., dong Zhang, X., and Li, H. Learning to search web pages with query-level loss functions. Tech. rep., 2006.

[24] Ristad, E. S., and Yianilos, P. N. Learning string-edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20*, 5 (May 1998), 522–532.

[25] Shalizi, C. *Advanced Data Analysis from an Elementary Point of View*.

[26] Weisstein, E. W. Logistic Function.

[27] wikipedia. Gradient Descent.

[28] wikipedia. Linear Regression.

[29] Xu, J., and Li, H. Adarank: A boosting algorithm for information retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2007), SIGIR '07, ACM, pp. 391–398.

[30] Yue, Y., Finley, T., Radlinski, F., and Joachims, T. A support vector method for optimizing average precision. In *SIGIR 2007: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Amsterdam, The Netherlands, July 23-27, 2007* (2007), pp. 271–278.